ECE 403: Senior Design II
Options Considered
Group 923

October 13th, 2009

Thomas Narvesen
Filipe Betzel
Zach Skalsky
Adam Hoffert

Introduction:

The project will cover the design and implementation of a custom microcontroller device. The microcontroller will consist of an open source microprocessor core implemented with three peripherals that the design team will build. Packet Digital LLC will be the customer for the end product, and will help to guide the design team in the design and construction of it.

Previous Work:

There are innumerous microcontrollers available in the market today. The scope of these types of devices is enormous. They range from simple 4 bit microprocessors which are used in toasters to high speed 64 bit DSP cores running complex digital image processing algorithms. Most microcontrollers are commercial packaged items with features set by the manufacturer. Even though these are often relatively cheap and fit most applications, the use of FPGAs as a basis for more customizable microcontrollers has increased. A number of open source or paid IP cores (semiconductor intellectual property core) have had a profound impact on the development of SoCs (system on chip), ASICs (application specific integrated circuit) and FPGAs (which our project is about). While our project is by no means unique in nature, we hope to use the tools and resources available today to create a FPGA based microcontroller with its own unique set of features and peripherals. Following is a list of some of the things that are available out there:

Commercial microcontrollers

The following Wikipedia page lists some of the most common of the thousands of devices available: http://en.wikipedia.org/wiki/List_of_common_microcontrollers. The one microcontroller most ECE students at NDSU may be familiar with is the PIC from Microchip Technology. For example the PIC 18F4620 has the following features: Harvard Architecture, 40 pins, 64 KB program memory, 3968 bytes of RAM, 1024 bytes of EEPROM data, comparator, ADC, USART, I2C, SPI, four timer modules, up to 5 PWM outputs, and up to 2 capture/compare. The commercial MSP430F123 from Texas Instruments, the core which our project will be based on, has: ultra low power operation, 8 KB flash memory, 256 bytes of RAM, runs at 8 Mhz, ADC, USART, watchdog timer, and analog comparator.

Full microcontrollers on FPGA

The following quote is taken from “FPGAs Are Everywhere – In Design, Test & Control”, RTC magazine: “A Gartner report shows that by 2010, more than 40 percent of all FPGA designs will contain an embedded microprocessor.” Therefore, embedding microprocessors into FPGAs is becoming increasingly more common. But since most projects that use microcontrollers on FPGAs are intellectual property of the company that designs them, it is hard to tell exactly what has been done so far. One application that is talked about on the article from mobiledevdesign.com entitled “FPGA signal processing for radar/sonar applications” is to use FPGAs not only for support functions on radar/sonar systems, but also to run optimized processing solutions that were previously based on proprietary math functions.

Microprocessor cores

A number of soft cores have been developed, especially by FPGA manufacturers. The Nios from Altera, the PicoBlaze from Xilinx, the open source Mico32 from Lattice, are just a few of the soft cores available. Opencore.org also has a number of open source microprocessor cores free for download and use. For this project we will not be designing the core itself, but we will rather use one of the open source cores available.

Microcontroller peripherals

This is the main part of our project since we will be designing these ourselves. Ideally we should incorporate something unique to any of the peripherals commonly found on microcontrollers. Some of the main peripherals common to most microcontrollers are: ADC, DAC, UART, I2C, SPI, Timers, PWM, Capture/Compare, LCD drivers, JTAG, internal oscillator, power management controls, watchdog timer, memory, and hundreds of variations of these and more. We will need to find the right combination of peripherals that will make our project useful and unique, and then research interface methods.

Design Options and Selected Approach:

Overview:

The project involves integrating components from different sources as well as our own work. We need an open source microprocessor core, three peripherals to integrate with it, software to write and test our design, one or more FPGA development board to test our design on, and an IDE to develop software to run on our device. We only have a limited amount of time and money to spend on our project, so we had to look closely at what we wanted to accomplish and what was available.

Options Considered: Microprocessor Core

The microprocessor core will be the heart and soul of our device. It is critical that we find a processor that is in a very mature and operational state, since the goal of the project is to develop peripherals for it. Any time we spend debugging or re-writing the processor core will take time away from other areas of our project.

We investigated several different microprocessors, both from Opencores.org and a commercial software micro from Lattice.

Lattice Mico8

Description:

The Lattice Mico8 is an 8 bit RTL microprocessor designed by Lattice Semiconductor.

Pros:

- Professionally designed and written, so bugs are less likely
- Open source, so we don't have to pay for it

- 25 pages of documentation, so we will have a good idea of how it works
- 8 bit architecture, so will be somewhat simpler than 16 or 32 bit processors

Cons:

- Must be modified to run on anything but a Lattice FPGA
- Code for micro must be written in assembly language using Lattice tools
- Design group has no experience using Lattice tools and equipment
- Design includes several features that we were thinking of implementing, features would have to be ignored or removed for our purposes
- Not open source in the true sense of the word, since it was developed by a company

OpenMSP430

Description:

The openMSP430 is an open source implementation of the Texas Instruments MSP430 that was found on Opencores.

Pros:

- Fairly good documentation
- MSP430 is a common processor, so we should be able to use TI tools for writing code
- Open source, so we won't have to worry about licensing at all
- 16 bit processor, so will be more powerful
- Can be used with a variety of different FPGAs

Cons:

- 16 bit, so it will be more complicated than 8 bit processors
- Contains some features that we thought about implementing ourselves, will have to remove them or find other peripherals to make
- Open source, so code may be slightly more buggy

PPX16

Description:

The PPX16 is another core that we found from Opencores. It is an implementation of the PIC16C55 or PIC16F84.

Pros:

- Design group is familiar with operation of PIC processors from former classes.
- 8 bit design will be easier to work with than 16 or 32 bit processors
- Open source, so no licensing worries
- Can be used with a variety of FPGAs

Cons:

- Little to no documentation about how the processor was designed
- Hasn't been updated since 2007, so some of the design may be different than the current processors

Approach for Microprocessor Core:

After evaluating the different processor cores, we decided to go with the openMSP430. We decided that because the documentation available on it was very good, and it is modeled off of a real world processor that has been around for a long time. The other processor modeled on real world hardware did not have much documentation, which would probably have made it harder to work with. We also thought that picking the Lattice core would limit our choices for development boards, and force us to use something that we didn't have any experience working with.

Options Considered: Peripherals

The peripherals are going to be what our project is all about. They will help interface our microprocessor core with other devices so that it can perform useful tasks. There are many different peripherals out there, but they are of differing difficulty to implement, and have different purposes. We need to choose three of them, so we are going to research at least five different peripherals.

I2C

I2C (Inter-Integrated Circuit) is a multi-master serial communication bus developed by Philips in the early 80's. It is used as a simple way to connect several low speed peripherals to one or multiple master controllers.

Advantages

- As of October 1, 2006 no licensing fees are required to implement the protocol
- Industry standard serial communication protocol
- Hundreds of compatible peripherals in the market
- Only two bus lines are required
- Lots of documentation available on the web
- No strict baud rate required

Disadvantages

- It is a very common feature on ICs, so it would not add uniqueness to our project
- Even though several devices can be connected to the bus, only one device can use the bus at a time
- Strict protocol specification, such as word and message size, stop and start bits, hardware addressing, etc
- Strict electrical specifications, especially for high speed or long distance transmission

SPI

Just like I2C, SPI (Serial Peripheral Interface) is a serial interface protocol for connecting peripherals to a master device, such as a microcontroller. It normally uses a 4-wire connection in full-duplex mode in single-master single-slave, or single-master multiple-slaves, configurations.

Advantages

- One of the simplest serial communication interfaces to implement
- No specifically defined protocol, making it easier to implement and allowing for variations
- One of the fastest synchronous serial data transfer interface, making it ideal for accessing memory, Ethernet controllers, ADC, and other high speed peripherals
- Full duplex mode allows for simultaneous send/receive

Disadvantages

- No hardware presence or data reception acknowledgment
- Supports only one master device
- Support for multiple slave devices is present, but becomes troublesome as number of devices increase
- Use of a minimal of 3-wires (4-wire for multiple slave support) requires more hardware resources than I2C
- It uses what is called a “de facto standard”, which means that there is no protocol standard agreed by an international committee. Even though variations in protocol can be easily accounted for in software, it requires additional effort when implementing SPI enabled third party devices

UART

UART stands for “Universal Asynchronous Receiver-Transmitter.” It is a standard that is used to take a byte of data and transmit it to another device serially. When the data arrives at the destination, another UART controller reassembles the byte. The standard is very common, and has a wide range of baud rates and options.

Pros:

- Very common standard, would allow our device to communicate with a lot of external devices

Cons:

- A rather complex standard: would take a lot of effort to make sure that our device meets all areas of the specifications
- Not very unique as almost all microcontrollers contain a UART for communication

PWM Module

A PWM module acts as a dedicated pulse width modulation driver. PWM could also be implemented in software using counters and turning ports on and off, but this is a rather inexact science. The number of other instructions in the software will change the loop time, and hence change the duty cycle and period of the PWM. If implemented in hardware, the PWM could be much more accurate as well as more predictable.

Pros:

- Normally implemented on motor control uCs, gives our product more flexibility
- Design would not be too complex, would be a good peripheral to start with
- Would allow uC to do other tasks while running a PWM device

Cons:

- Very specific piece of hardware, would not have that much flexibility
- Might be a little too simple for the scope of our project
- Already can be implemented using TimerA in the openMSP430

LCD Driver

An LCD driver would integrate something into our uC that would make it quite a bit more unique. This can also be done in software, but updating the LCD is very processor intensive. Almost all of the processor time is spent sending single characters to the LCD, and it needs to wait in between sending the characters. If we could find a solution to write to the LCD without needing to have the processor kill time and wait, it would speed up applications that use the LCD considerably.

Pros:

- We couldn't find a dedicated driver for a microcontroller anywhere, so this is fairly unique.
- Would be interesting to implement, since we would have to come up with our own requirements and design for the peripheral.
- Helps mix things up a bit; many of our other options are simply communication peripherals that have been done time and time again

Cons:

- We would have to come up with our own design and requirements, which would be time intensive and error prone
- Not a lot of applications use LCD screens, so this option would add cost to our project for a relatively small market

Wishbone

Wishbone is a type of communication bus that allows devices on the same chip to communicate with each other. The Wishbone standard was designed to be flexible so it could work with many different IP cores, but provide a framework to increase compatibility and promote IP reuse. It comes in 8, 16, 32 and 64 bit buses and uses parallel communication.

Pros:

- Would help our controller integrate with other peripherals or cores by providing a standard communication scheme

Cons:

- Not very useful for integrating our controller with outside devices.

Pulse Accumulator

A pulse accumulator is an event counter. Each time an event occurs the pulse accumulator increments a register variable, thus recording the number of times this event occurred.

Advantages

- Many different applications, such as receiving signals from optical or magnetic encoders
- Should be fairly straight forward to implement

Disadvantages

- Already included as part of TimerA in our core, so designing another one wouldn't add much to the project
- Not unique among microcontrollers

Power Management Module

Power Management Modules come in a variety of forms, from simple blocks of code designed to manually turn off certain parts of the device, to specific chips that control all aspects of power consumption in a system. In this design we will try to implement a simple built in power management module that can monitor the state of the different parts of the microcontroller and automatically switch them from active to low power state, or vice-versa.

Advantages

- Power management has been increasingly more of a concerned among microcontrollers, especially because of the emergence of many consumer battery powered devices
- MSP430 family from Texas Instruments is advertised as a low power microcontroller, so it would only make sense to keep that in mind
- Opportunities for learning basics of power management on digital devices
- Flexibility of approaches

Disadvantages

- It can become quite complex depending on the performance we want to achieve and methods we want to use
- It may be hard to measure and demonstrate the results on a development board

Approach for Peripherals:

Looking over all the peripherals carefully and picking the right three was important because it is the peripherals that will give the microcontroller its defining characteristics. The first one we chose was the UART. Its simplicity should make it a good module to help get us acquainted with the task of designing a device. It also gives us a standard for communicating with outside devices. The second device we picked was the LCD drive. Writing code in software for a LCD can be very tedious but with a LCD drive we would make that process simple and fast. An LCD drive offers our microcontroller a unique style and something other than a communication device to work with. The final peripheral we chose was the SMBus at the recommendation of our project advisor. The SMBus commonly communicates with temperature, fan and voltage sensors. Combining all three of these peripherals we should be able to find several different purposes for the microcontroller.

Options Considered: Dev Boards and Software

Altera

DE1

Pros:

- Uses the familiar Quartus II software which is free
- It has a simple monitoring program
- Other features include SRAM, SDRAM, and flash memory chips.
- This is very user friendly and a decent price compared to the other boards.
- This will work with lattice core.

Cons:

- This board doesn't have all the extra's that the DE2 has
- Has I2C buss which is something we talked about designing on our own

DE2

Pros:

- This board has more toggle switches, LED's, and 7 segment displays than DE1
- It seems to be well documented

Cons:

- This will cost about a hundred dollars more than the DE1.
- Contains a lot of extra components that we probably won't use

Xilinx

Pros:

- Has push buttons, dip switches and other necessary inputs
- Onboard JTAG configuration circuitry

Cons:

- Cost is a major concern with this board, as it is 495 dollars
- No experience working with Xilinx hardware or software

Lattice

Pros:

- Has several inputs and outputs for various pins
- Would be able to run the Lattice 8 bit processor core

Cons:

- The development board is very sparse, with not very many extra features
- Priced at $295, so definitely within our budget
- No experience with Lattice hardware or software

Approach for Development Board and Software:

For the development board and dev software, we decided to go with Altera. The DE1 and DE2 are very user friendly, and we have some experience working with them before. In addition, our project adviser is also proficient in their use, and the Quartus software supplied with them is familiar to us as well. These options made the choice fairly obvious in this section.

Software Packages

Our choice of software is rather limited due to the expense required for obtaining such software. For simulation of our RTL, we will be using the design tools available in the computer lab. Our advisor has a lot of experience using the Cadence tools, and they are far superior to anything else that we could get for free. In addition, using them will help us to gain real world experience, because many companies, including our sponsor, use them. For writing and compiling the operational code for our processor, we will need to use the TI IDE for the MSP430. We will be able to write our code in C, and the compiler will do the rest. It wouldn't make any sense to try and find another compiler elsewhere when this one will work with little or no modification.

Other Options

As for other options for our project, we added a couple of lines to our budget for a peripheral board and some components. We aren't sure if we are going to need these or not, but at a suggestion from our advisor we put them in just in case. It is possible that some of our peripherals will need external circuitry to interface them with other outside devices, and we wanted to keep our options open.

Budget

Part	Price	Academic Price	QTY	Total
Altera DE1 Board	$150	$125	1	$125
Altera DE2 Board	$495	$269	1	$269
Internal PCB	$0	$0	2	0
External PCB estimate	$50		1	$50
Parts estimate	$30		1	$30
Development Software	$0	$0	1	0
Shipping	$10	$10	4	$40
TOTAL				$514

Summary

Our project will consist of several tasks. We will be using a free soft processor from Opencores.org and interfacing three peripherals that we will design and build ourselves. The three peripherals will be a UART module, an SMBUS and an LCD controller. The peripherals will be written in Verilog and simulated using the Cadence design tools available in the department computer lab. After design, coding and simulation, the finished software microcontroller will be loaded onto an FPGA development board to verify operation of the device. Software for the microcontroller will be designed and written using the tools available from Texas Instruments. An application will be used to test the functionality and demonstrate the usefulness of our project. After the design has been verified and shown to work, we will attempt to develop our design into an integrated circuit using the Cadence extraction and layout tools. Since we aren't sure how long the first steps will take, the extraction and layout will only be done if we finish the other tasks far ahead of schedule. Carrying out this project will help the group develop a sense of what it takes to develop and implement a new semiconductor device, and hopefully better prepare us for a job in today's rocky market.